

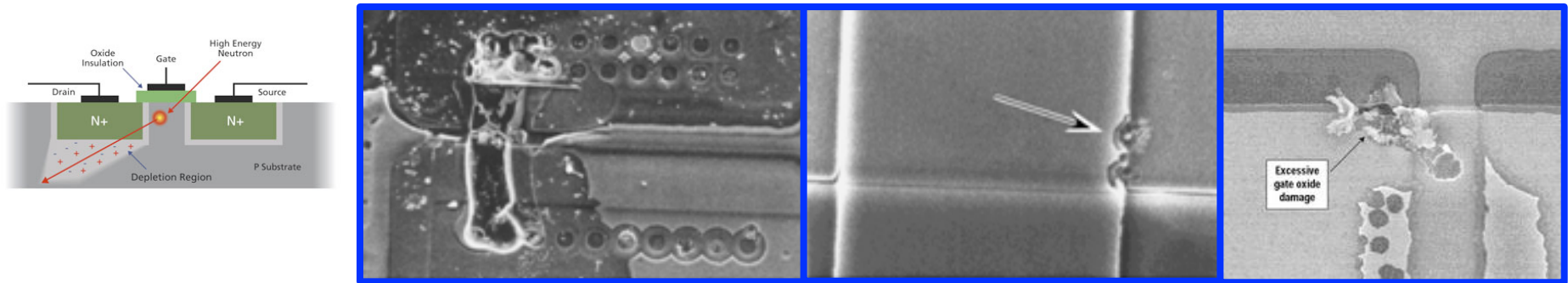
# Resilient Algorithms and Computing Models

Franck Cappello  
Argonne/MCS Resilience lead  
ANL, UIUC



# Type of errors

- **Hard error:** An irreversible change in operation that is typically associated with permanent damage to one or more elements of a device or circuit (e.g., gate oxide rupture, etc.).



metal melt, gate oxide damage

- **Soft error** (transient errors): An erroneous output signal from a latch or memory cell that can be corrected by performing one or more normal functions of the device containing the latch or memory cell:
  - Cause: Alpha particles from package decay, Cosmic rays creating energetic neutrons
  - Soft errors can occur on transmission lines, in digital logic, processor pipeline, etc.
  - Soft error rate (SER).
  - **BW (1.5PB of memory): 1M memory errors in 261 days → 1 every 4s**

<http://www.jedec.org>

# Silent Data Corruptions?

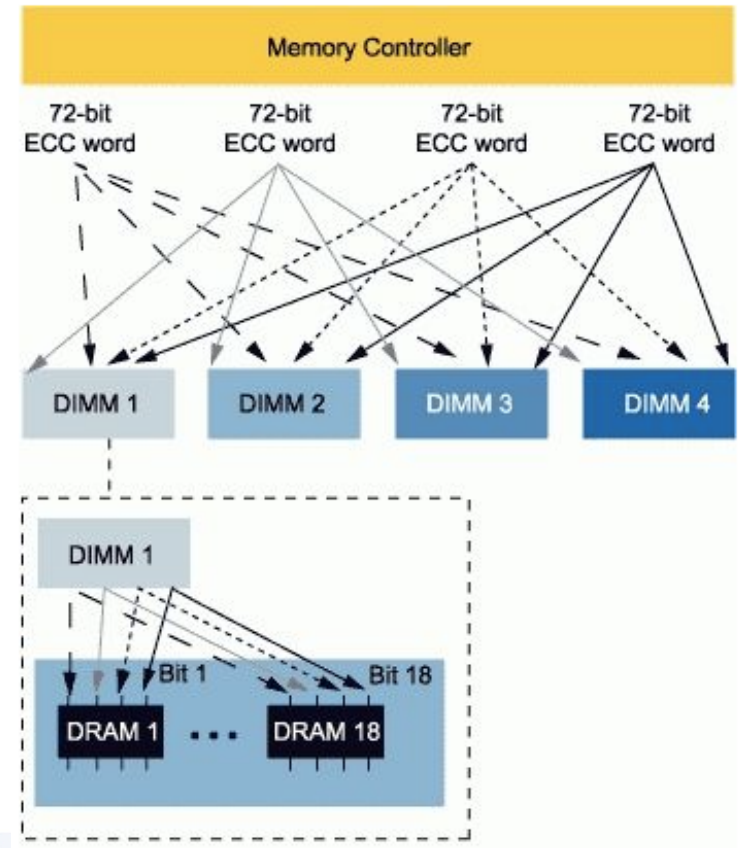
Errors affecting data become Silent Data Corruptions if they are not detected. Some errors in control flow also lead to SDCs.

Silent Data Corruptions may be absorbed (no effect), may lead to process crashes (noticeable effect) or may lead to wrong results (silent effect).

Main source of Errors:

- Memory (but Chipkill and ECC reduce the probability of SDCs to negligible)
- BW: 28 uncorrectable errors in 261 days**
- Toward exascale (Latches, Flip flops in processor pipelines and control structures)

Classic Hardware techniques



# Exascale Projections (from ICIS report 2014)

FIT: number of failures that can be expected in one [billion](#) ( $10^9$ ) device-hours of operation

	Array interleaving and SECDED (Baseline)					
	DCE [FIT]		DUE [FIT]		SE [FIT]	
	45 nm	11 nm	45 nm	11 nm	45 nm	11 nm
Arrays	5000	100,000	50	20,000	1	1000
Scattered latches	200	4000	N/A	N/A	20	400
Combinational logic	20	400	N/A	N/A	0	4
DRAM	50	50	0.5	0.5	0.005	0.005
Total	1000–5000	100,000	10–100	5000–20,000	10–50	500–5000

~1000 FIT per processor → 1M processors → 1B FIT → 1 SE per hour

	Array interleaving and SECDED + latch parity					
	DCE [FIT]		DUE [FIT]		SE [FIT]	
	45 nm	11 nm	45 nm	11 nm	45 nm	11 nm
Arrays	5000	100,000	50	1000	1	5
Scattered latches	200	4000	20	400	0.01	0.5
Combinational logic	20	400	N/A	N/A	0.2	5
DRAM	0	0	0.1	0.0	0.1	0.001
Total	1500–6500	100,000	25–100	2000–10,000	1	5–20

Array interleaving and SECDED + latch parity

(45 nm overhead ~10%; 11 nm overhead: ~20% area and ~25% power)

May not happen due to lack of market

# This is not only for exascale: case 1

## Los Alamos experiment (Credit: Nathan DeBardeleben)

- NVIDIA Tesla M2090 GPGPUs (Fermi)
- CUDA 5.0
- 40,171 single GPU runs of HPL : 16,738 total hours
- 1624 bad residual (test at the end of HPL): **4% of the runs**
- **Only 1.6%** of those bad residuals reported in event log
- 11,000 runs of HPL on 2 K20 nodes (Kepler)
- **6 Bad residuals...**
- **And...HPL residual check is a bad test for detecting data corruptions (corruptions on the right part of the mantissa will be unnoticed!)**

## Argonne experiment (Credit: Leonardo Bautista Gomez)

- No corruption on 9000 runs of 20K HPL on M2050 (Fermi)

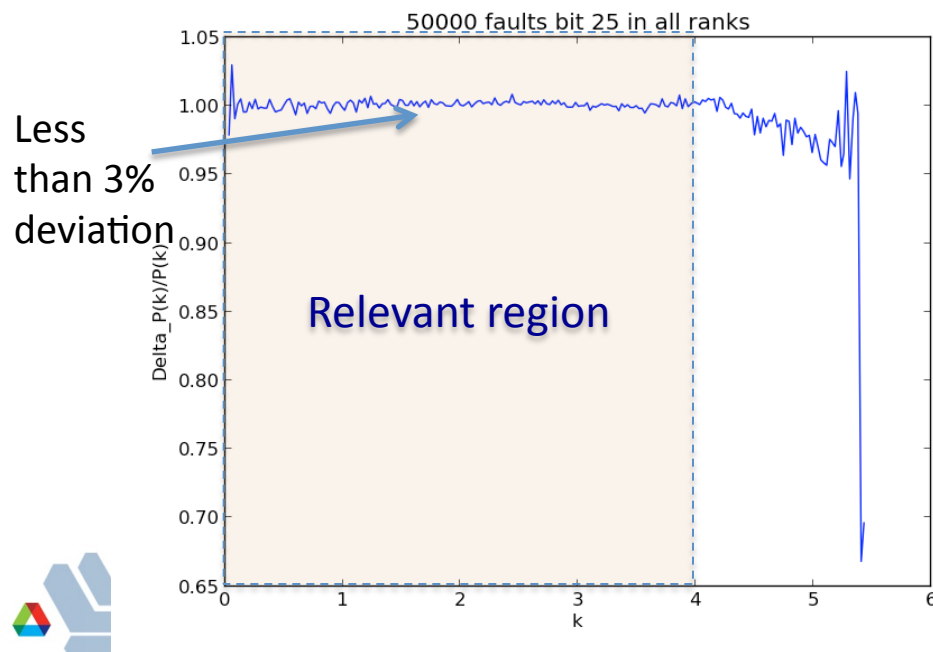


# Sanity checks may not suffice: case 2

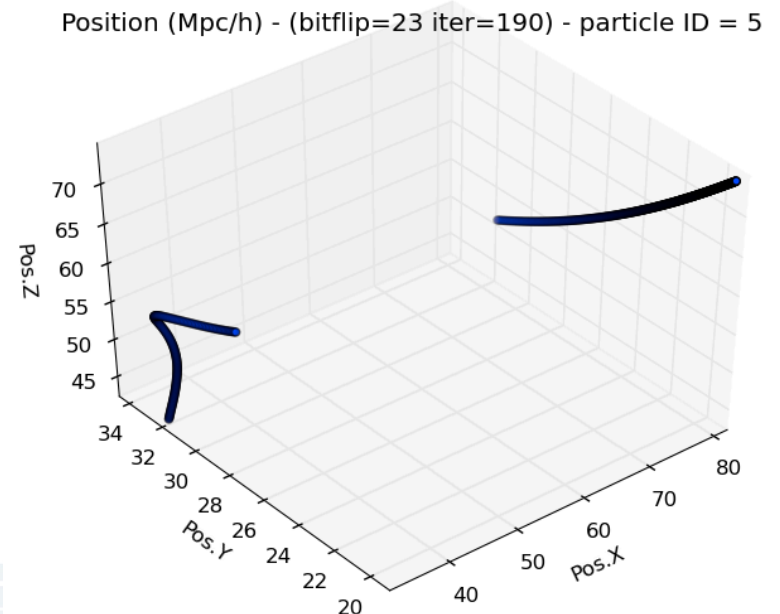
## HACC (Cosmology: Particle position from gravity, FFTs, density fields)

- Issues with Memory DIMM, Issues with I/O, Bugs
- The power spectrum represents the amplitude of variations on different mass scales. It is also used as high level checking of the computation.
- Effect of bit flips in particle position are not observable on the power spectrum BUT the result may be corrupted.
- Bit flips in FFTs or density field may lead to more noticeable effects

Power spectrum



Particle trajectory



# Optimization algorithms sensitivity: case 3

Optimization algorithms iteratively reduce the error in the initial guess to reach the intended solution.

→ The seem to be resilient to SDC: Yes, but to a certain extend!

Checked the Hartree–Fock method (optimization algorithms).

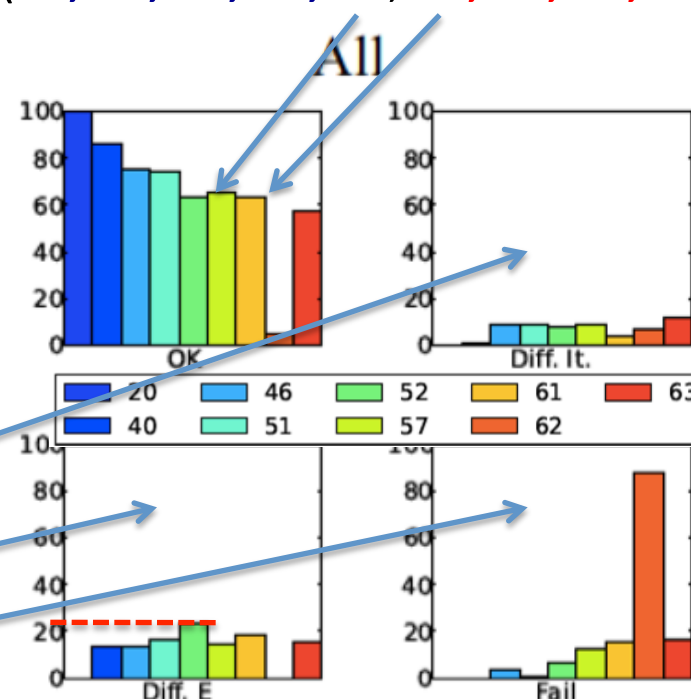
- Inject a single bit flip in floating point #, in bit (20, 40, 46, 51, 52, 57, 61, 62, 63)
- Randomly corrupted data elements
- At least one SDC in each execution
- Obtained statistical results

Percentage of executions that converge to  
Correct solution or suffer from SDCs

**Converge but need more iterations**

**Converge to different energy**

**Fail to converge (stop abruptly)**



Van Dam, H. J. J., Vishnu, A., and de Jong, W. A. (2013). A case for soft error detection and correction in computational chemistry. *Journal of Chemical Theory and Computation*, 9(9):3995{4005.

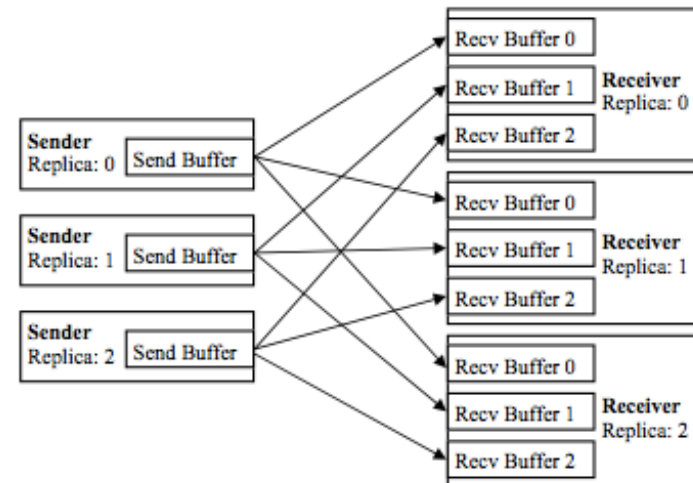
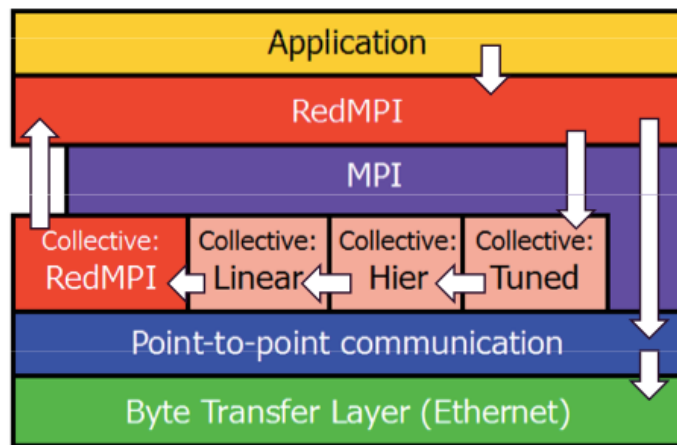


# Detection from classic replication

Replicate execution and compare execution state

Ex: RedMPI: compares content of messages

coming from MPI process replicas



LAMMPS INPUT CHUTE.SCALED

Size	1x [sec]	2x [sec]	3x [sec]	2x OV	3x OV
128	137.50	138.38	139.01	0.6%	1.1%
256	138.26	140.43	140.00	1.6%	1.3%
512	139.19	140.22	140.67	0.7%	1.1%

- Low performance overhead
- A nice approach if you can afford 2x or 3x the resources and energy.





# Detection from Numerical Algorithms: ABFT

Huang and Abraham, proposed the ABFT (1984) to detect and correct errors in some matrix operations. Use row and column checksums and property of linear algebra: Operation on checksummed inputs produce checksummed result that is checked for detection

To support  $f$  failures,  $f$  checksums  $y_i$  are computed

$$\begin{cases} y_1 = a_{11}x_1 + \dots + a_{1p}x_p, \\ \vdots \\ y_f = a_{f1}x_1 + \dots + a_{fp}x_p. \end{cases}$$

Works for many Linear Algebra operations:

Matrix Multiplication:  $A * B = C \rightarrow A_c * B_r = C_f$

LU Decomposition:  $C = L * U \rightarrow C_f = L_c * U_r$

Addition:  $A + B = C \rightarrow A_f + B_f = C_f$

Scalar Multiplication:  $c * A_f = (c * A)_f$

Transpose:  $A_f^T = (A^T)_f$

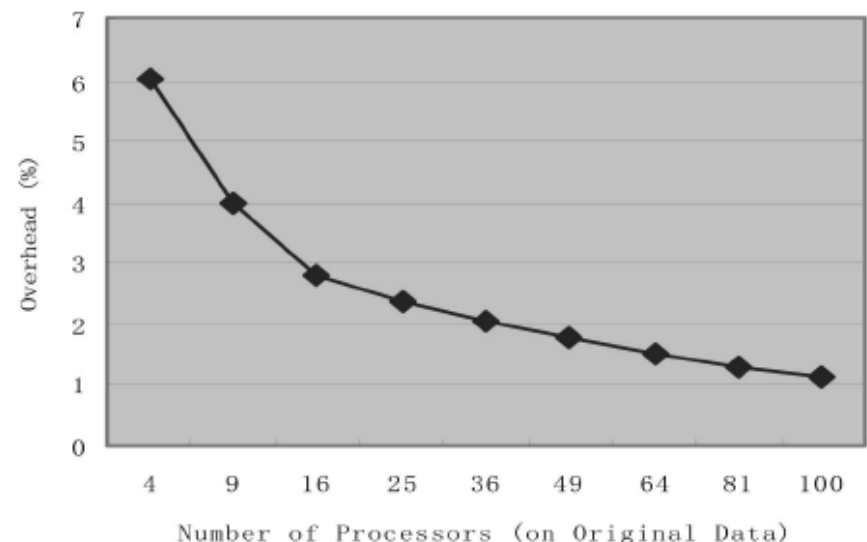
Cholesky factorization & QR factorization

Performance is quite good

Figure presents the overhead for computing the matrix product on encoded matrices.

The time to compute the checksum should be added as well.

Overhead for Performing Computations on Encoded Matrices



# Generalization of ABFT

Latest results in that domain

## Online ABFT for Krylov subspace iterative methods

simple verification of orthogonality and residual

→ tested on BiCG, and Lanczos biorthogonalization, PCG

→ Works also for GM-RES, Arnoldi  
(eigenvalue problems), etc.

Zizhong Chen. 2013. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings ACM PPOPP '13*

## Online ABFT for LU, QR and Cholesky

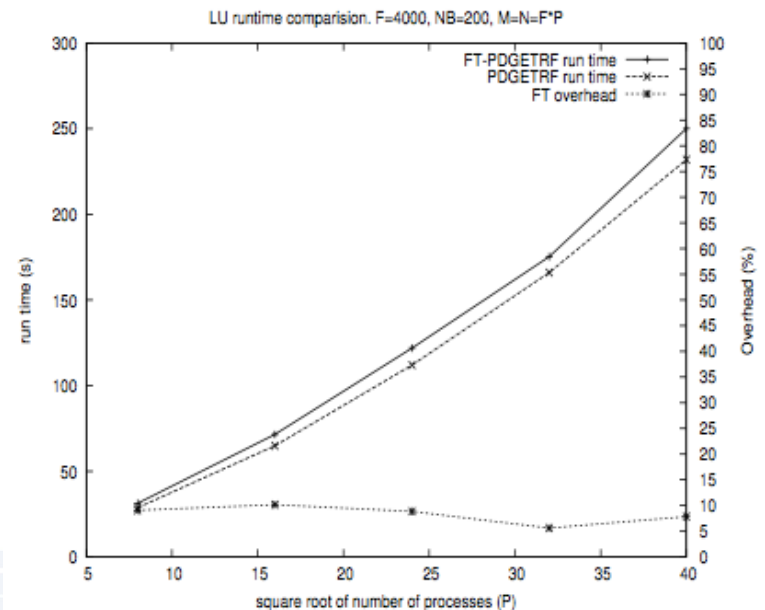
→ Overhead stays <10% (fault free situation)  
(Stampede, weak scaling)

Panruo Wu , Zizhong Chen , FT-ScaLAPACK: Correcting Soft Errors On-Line for ScaLAPACK Cholesky, QR, and LU Factorization Routines, Proceedings of ACM HPDC 2014

```

1 : Compute  $r^{(0)} = b - Ax^{(0)}, z^{(0)} = M^{-1}r^{(0)}, p^{(0)} = z^{(0)}$ ,
    and  $\rho_0 = r^{(0)T} z^{(0)}$  for some initial guess  $x^{(0)}$ 
2 : checkpoint:  $A, M$ , and  $b$ 
3 : for  $i = 0, 1, \dots$ 
4 :   if ( $i > 0$ ) and ( $i \% d = 0$ )
5 :     if ( $\frac{p^{(i+1)T} q^{(i)}}{\|p^{(i+1)}\| \|q^{(i)}\|} > 10^{-10}$ 
        or  $\frac{\|r^{(i+1)} + Ax^{(i+1)} - b\|}{\|b\| \|A\|} > 10^{-10}$ )
6 :       recover:  $A, M, b, i, \rho_i,$ 
         $p^{(i)}, x^{(i)},$  and  $r^{(i)}$ .
7 :     else if ( $i \% (cd) = 0$ )
8 :       checkpoint:  $i, \rho_i, p^{(i)},$  and  $x^{(i)}$ 
9 :     endif
10 :   endif
11 :    $q^{(i)} = Ap^{(i)}$ 
12 :    $\alpha_i = \rho_i / p^{(i)T} q^{(i)}$ 
13 :    $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$ 
14 :    $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$ 
15 :   solve  $Mz^{(i+1)} = r^{(i+1)}$ , where  $M = M^T$ 
16 :    $\rho_{i+1} = r^{(i+1)T} z^{(i+1)}$ 
17 :    $\beta_i = \rho_{i+1} / \rho_i$ 
18 :    $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$ 
19 :   check convergence; continue if necessary
20 : end

```



# Low cost techniques

Previous techniques try to detect exhaustively errors

→ There is no optimization of a trade-off between detection and cost (overhead in time, resource, energy,)

What about low cost techniques that would detect most of the errors but cannot guarantee detecting all errors?

→ **Approximate detection**

→ The important notions are:

→ **Coverage:** % of detected SDCs

→ **Overhead:** % more time, resource, energy (checker+false alerts)

Important: in the next slides only 1 method reports Coverage & overhead



# Low Cost Hardware Detection

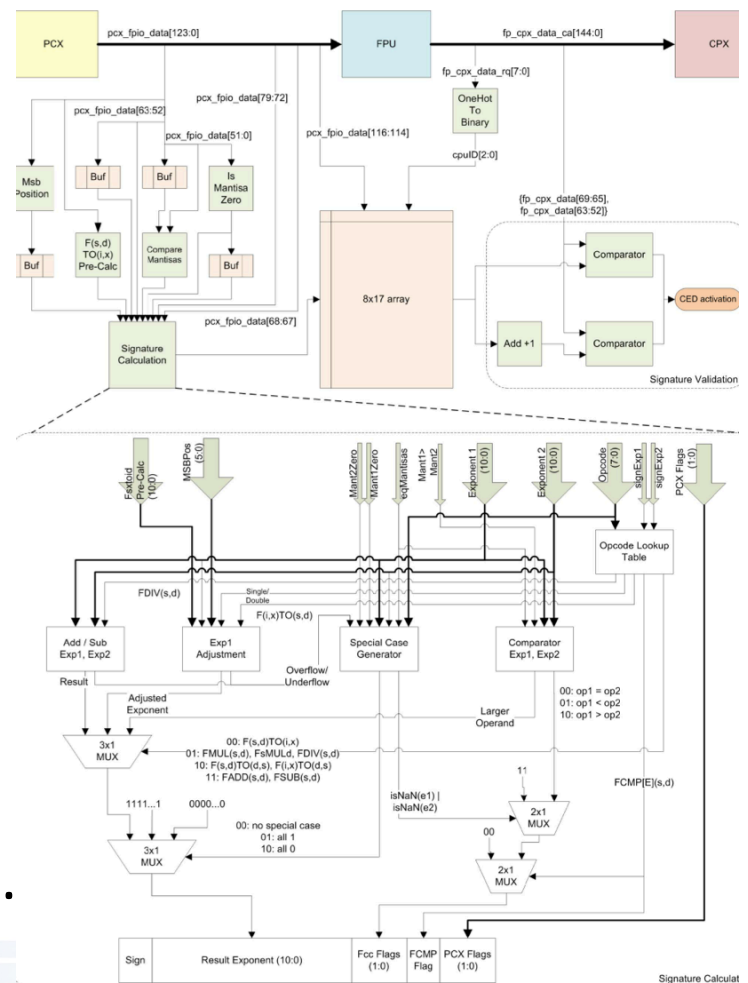
Example: M. Maniatakos et al., Low-Cost Concurrent Error Detection for Floating-Point Unit (FPU) Controllers (2013)

Conjecture: Error in the control logic will lead to extensive data path corruption, which will propagate to the exponent part of the floating-point representation.

→ Check the exponent  
+ base-15 residue code for the fraction

SDC detection solution for the entire FPU:

- **Coverage of 94.1** percent of all errors
- Area cost of 16.32 percent of the FPU size.



# Low Cost Algorithm Specific Detection

A first approach is to develop ad-hoc detection approaches

## Case 3: Hartree–Fock optimization algorithm:

- Mechanisms for detecting and correcting soft errors in various data structures
- Replicated and distributed data structures (geometry and basis set objects)
- Checksum (MD5) for replicated data structures compared during execution
- For distributed data structures:

Detection from numerical assertions:

- Conditions or bounds on the valid values
- Exact Condition for Detecting Orbital Orthonormality Violations, etc.

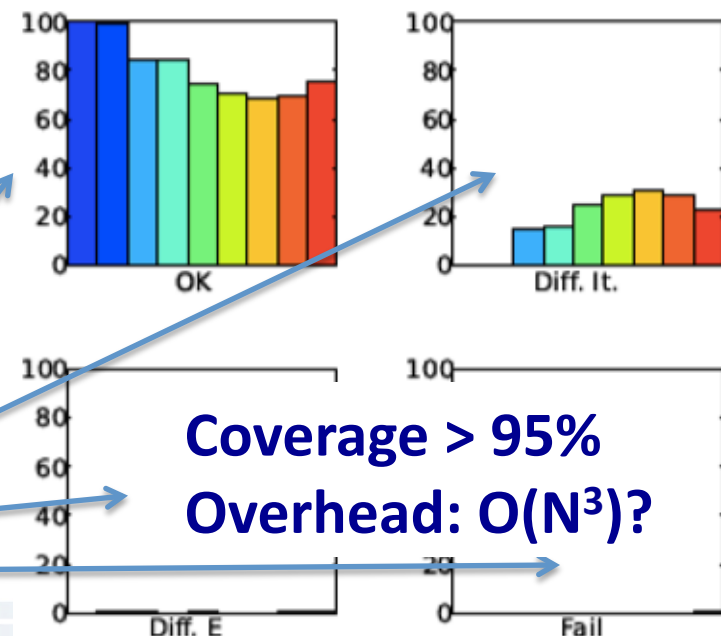
Correction by replacing value.

**Converge as in SDC free situation**

**Converge but need more iterations**

**Converge to different energy**

**Fail to converge (stop abruptly)**



**Coverage > 95%**  
**Overhead:  $O(N^3)$ ?**



# Low Cost Detection from Auxiliary Scheme

Scope: time dependent, initial value problems.

- Compare the solution given by a primary time stepping scheme to the one given by an auxiliary checking scheme at every time step (same initial conditions)
- The auxiliary scheme « restarts » from the primary intermediate solution at each time step (not a closed loop scheme).
- Detection from a sequence of the norm of the difference between prim. and aux. solutions

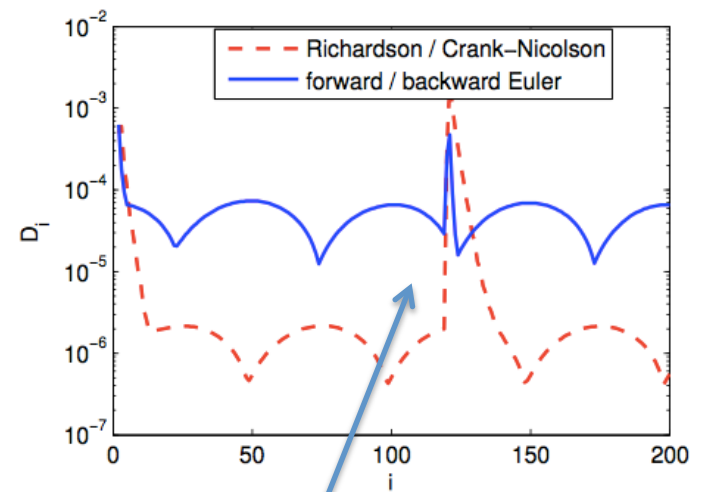
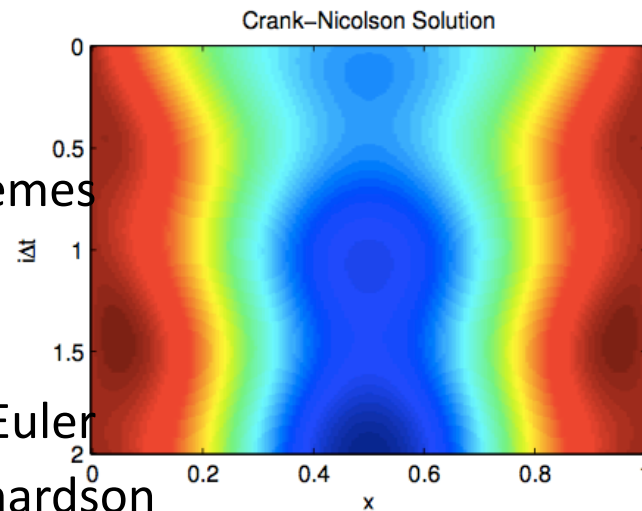
Example:

-finite difference schemes  
for PDEs

-Heat equation:

--Backward/forward Euler

--Crank-Nicolson/Richardson



Detection of the norm of the difference between the two solutions

- For now, no localization of the SDC

Benson, Austin R., Sven Schmit, and Robert Schreiber. "Silent error detection in numerical time-stepping schemes." International Journal of High Performance Computing Applications (2014).

# Conclusion

The community will have to deal with SDCs (with a high probability)

The main problem is detection (numerous recovery approaches)

Classic techniques from distributed systems (replication) is not directly applicable (too expensive in resource, energy overheads)

Other approaches like ABFT exist for linear algebra, FFTs, etc.

For other kernels, one approach is **approximate detection**:

- Hardware level, algorithmic level, scheme level and different scales.

**The precise understanding of the applicability of the approximate detection techniques is an open problem**







Questions?

